VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2019 Pavol Majchrák

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student:	**Pavol Majchrák**
Studijní program:	B2647 Informační a komunikační technologie
Studijní obor:	2612R025 Informatika a výpočetní technika
Téma:	Absolvování individuální odborné praxe Individual Professional Practice in the Company
Jazyk vypracování:	čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: IDC CEMA, s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Tomáš Fabián, Ph.D.**

Konzultant bakalářské práce: Ing. Roman Krpel

Datum zadání: 01.09.2018
Datum odevzdání: 30.04.2019

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

V Ostrave 23. dubna 2019 ..... Majchrák .....

Súhlasím so zverejnením tejto bakalárskej práce podľa požiadaviek čl . 26, odst. 9 Študijného a skúšobného poriadku pre štúdium v bakalárskych programoch VŠB-TU Ostrava.

V Ostrave 23. dubna 2019

IDC CEMA s.r.o.
Malé náměstí 13, 110 00 Praha 1
IČ: 26482347 DIČ: CZ26482347

Rád by som sa na tomto mieste poďakoval mojim kolegom z práce, ktorý boli vždy ochotní mi pomôcť a vysvetliť hocijaký problém. Poďakovanie patrí aj môjmu konzultantovi Ing. Romanovi Krpelovi, ktorý mi zastrešil odbornú prax. V neposlednom rade patrí poďakovanie pánovi Ing. Tomášovi Fabiánovi, Ph.D. za pomoc pri písaní bakalárskej práce.

Abstrakt

Táto bakalárska práca popisuje absolvovanie odbornej praxe vo firme International Data Corporation. Na začiatku práce je opísané zameranie firmy. Následne je popis môjho zaradenia vo firme, projektu a metodiky práce. V tretej kapitole sú stručne opísané technológie, ktoré som najčastejšie používal. V nasledujúcej kapitole je rozobratá časť úloh, ktoré som počas praxe riešil. Táto kapitola sa snaží stručne popísať analýzu a implementáciu jednotlivých úloh. V závere hodnotím aké prínosy ma pre mňa absolvovanie tejto odbornej praxe a taktiež využiteľnosť vedomostí, ktoré som získal v mojom doterajšom štúdiu.

Kľúčové slová: C#, WPF, MVVM, SQLite, XAML, IDC, odborná prax

Abstract

This bachelor thesis describes my individual practice in the company called International Data Corporation. The focus of the company is mentioned at the beginning of the work. It continues with description of my inclusion, project and work methodology in the company. The most used technologies are mentioned in the third chapter. The next chapter includes analysis and implementation of the most important tasks, which I have solved during my practice. Finally, I evaluate what benefits my practice has and how useful the knowledge I have gained in my previous study.

Key Words: C# , WPF, MVVM, SQLite, XAML, IDC, professional practice

Obsah

Zoznam použitých skratiek a symbolov

IDC	–	International Data Corporation
PO	–	Product owner
UX	–	User Experience
WPF	–	Window Presentation Foundation
MVVM	–	Model-View-ViewModel
TFS	–	Team Foundation Server
CLR	–	Common Language Runtime
XAML	–	Extensible Application Markup Language
HTTP	–	Hypertext Transfer Protocol
UDP	–	User Datagram Protocol
TCP	–	Transmission Control Protocol
GELF	–	Graylog Extended Log Format
JSON	–	JavaScript Object Notation
REST	–	Representational State Transfer
API	–	Application Programming Interfaces
QA	–	Quality Assurance

Zoznam obrázkov

Zoznam výpisov zdrojového kódu

1 Úvod

Moja bakalárska prax prebiehala vo firme IDC, ktorá sa zaoberá najmä analýzou IT trhu. V tejto firme som nastúpil na pozíciu aplikačný vývojár. Bol som priradený do tímu vyvíjajúceho desktopovú aplikáciu, ktorú používajú analytici na zber dát.

Hlavný dôvod pre výber praxe bolo získať skúsenosti a vyskúšať si ako prebieha vývoj reálnych aplikácií. Taktiež som chcel zužitkovať moje znalosti z doterajšieho štúdia pri implementácii aplikácie, ktorá pomáha pri práci iným ľudom.

Firmu IDC som si vybral najmä kvôli možnosti pracovať v tíme so skúsenými vývojármi. Prax sa mi podarilo nájsť na konci druhého ročníka, takže som začal pracovať už počas letných prázdnin. Počas praxe som získaval každý deň nové poznatky a skúsenosti. Vďaka code review procesu, mi vždy skúsený kolegovia pridali pripomienky, čo by v kóde spravili lepšie, podľa čoho som vždy kód prepísal. Na základe tohto procesu som sa naučil používať nové funkcionality jazyka a hlavne písať kvalitnejší kód. Pre rôznorodé požiadavky od product ownera som sa stretával stále s novými funkcionalitami aplikácie a najmä rôznymi oblasťami vývoja ako je napríklad užívateľské rozhranie, komunikácia so serverom, logovanie, používanie rôznych komponent na zobrazovanie dát a mnoho ďalšieho.

Počas praxe som chcel získať prehľad ako funguje vytváranie skutočných aplikácii. Vyskúšať prácu v tíme, zároveň zistiť ako prebieha celý proces od požiadaviek cez analýzu, návrh , implementáciu až po nasadenie pre používateľov. Ďalej som chcel získať prehľad o technológiách a metodikách , ktoré sa používajú pri vývoji moderných aplikácií. Taktiež som od praxe očakával získanie potrebných skúsenosti, ktoré mi zvýšia hodnotu na trhu práce.

2 Popis firmy a pracovnej pozície

2.1 Informácie o firme

Internation Data Corporation je popredný poskytovateľ poradenských služieb a informácií o vývoji trhu, predovšetkým v oblasti informačných technológií a telekomunikácii. V IDC pracuje viac ako 1100 analytikov po celom svete a analyzujú dáta z viac než 110 krajín[1]. V Čechách má IDC dve pobočky. Jedna je v Prahe a druhá v Ostrave. Tieto dve pobočky sú zamerané najmä na vývoj a údržbu aplikácií, ktoré používajú analytici.

2.2 Ferda

Ferda je aplikácia vyvinutá pre interné firemné použitie. Túto aplikáciu používa k analýze trhu, modelovaniu a predpovedaniu budúcich trendov na IT trhu vyše 1100 analytikov po celom svete. Ferda na serveroch uchováva terabajty dát z celého sveta. Analytici používajú desktopovú aplikáciu, v ktorej si navolia s akými dátami chcú pracovať a aplikácia stiahne potrebné dáta zo serveru. Tieto dáta sa môžu v aplikácií upravovať buď vo forme tabuliek alebo grafoch. Ďalej umožňuje vkladať nové dáta napríklad z Excel dokumentov alebo export dát do rôznych formátov. Pri zmenách dát umožňuje rôzne druhy balancingu a množstvo ďalšieho. Z programátorského hľadiska je to celkom náročný projekt, pretože sa pracuje s množstvom dát a je potreba zachovať čo najväčšiu rýchlosť[2].

Momentálne sa vyvíja druhá verzia aplikácie kde frontend je postavený na .NET frameworku ako WPF aplikácia. Väčšina komunikácie medzi klientskou častou a serverom prebieha prostredníctvom RabbitMQ. Na lokálne ukladanie dát používa aplikácia SQLite databázu.

Aplikácia má štyri verzie, stabilná produkčná verzia sa vydáva každé tri mesiace, pred vydaním tejto verzie prebieha vždy 3-5 dňové testovanie a opravovanie chýb. Do Preview tejto verzie sa pravidelné každé dva týždne pridávajú nové funkcionality. Ďalšie dve verzie aplikácie sú určené na testovanie a vývoj.

2.3 Pracovná pozícia

Do firmy som nastúpil na pozíciu aplikačný vývojár na part time. Bol som priradený do tímu v Ostrave, ktorý vyvíja frontendovu časť na aplikácií Ferda. V tomto tíme som spolupracoval so štyrmi skúseným vývojármi a jedným testerom. Túto aplikáciu taktiež vyvíja druhý frontend tím v Prahe a ďalšie dva backend tímy. Aplikácia ma jedného PO a jedného UX designera a jedného support specialistu. Pre riadenie projektu sa používa SCRUM metodika.

Prvé dva dni v práci som absolvoval viacero stretnutí s kolegami, ktorí mi vysvetľovali základné údaje o aplikácií, či už z užívateľského pohľadu ale taktiež jednoduchý architektonicky náhľad. Súčasne mi bol vysvetlený spôsob verzovania aplikácie, na čo sa používá Git a TFS. Potom som začal spolupracovať s tímom na aktuálne riešených úlohách.

Počas praxe som mal možnosť podieľať sa okrem programovania jednotlivých úloh aj na analýze a investigovaní nových požiadaviek. Taktiež som mal možnosť vyjadrovať sa k návrhu jednotlivých riešení. Pri implementácii množstva nových funkcionalít som pokrýval svoj kód Unit testami. Vyskúšal som vytvoriť aj pár integračných testov. Na pár hodín som skúsil aj manuálne testovanie aplikácie podľa pripraveného scenáru. Pri vývoji užívateľského rozhrania som bol často v kontakte s UI designerom.

2.4 Používané nástroje

Na vývoj aplikácie sa používa najnovšie Visual Studio. Na zjednodušenie práce sa používa ReSharper ako rozširujúci balík pre Visual Studio. Ako verzovací systém je použitý Git spolu s Team Foundation Server (TFS). Komunikácia medzi členmi tímu prebieha pomocou Microsoft Teams. Na projektové riadenia slúži Jira.

3 Použité technológie

Počas praxe som sa podieľal na vývoji desktopovej aplikácie. Táto aplikácia je postavená na Windows Presentation Foundation platforme , ktorá tvorí podmnožinu .NET Frameworku. Vývoj prebieha v programovacom jazyku C# a na oddelenie logiky od užívateľského rozhrania sa vyžíva návrhový vzor MVVM. Aplikácia vo väčšej miere prenáša dáta medzi klientskou a serverovou časťou vo forme SQLite súborov. Na správu verzií sa používa Git. Keďže som sa počas doterajšieho štúdia stretol s týmito technológiami len okrajovo, musel som sa ich naučiť samostatne.

3.1 Microsoft .NET Framework

.NET Framework je bezplatná platforma vyvinutá spoločnosťou Microsoft. Táto platforma je určená pre vývoj rôznych typov aplikácii. Obsahuje množstvo knižníc na vývoj pre web, desktopy, mobily, hry a taktiež IoT. Ďalšou výhodou je možnosť programovať vo viacerých programovacích jazykoch, napríklad Visual Basic, F# a C#. Framework obsahuje okrem knižníc aj samotné behové prostredie (CLR - Common Language Runtime), ktoré zabezpečuje beh a kompilovanie aplikácií. Prvá verzia bola vydaná v roku 2002 pod názvom .NET Framework 1.0. Momentálne najnovšia verzia je .NET Framework 4.7.2 [3].

3.2 C#

C# je objektovo-orientovaný programovací jazyk vyvinutý spoločnosťou Microsoft. Umožňuje vyvíjanie robustných aplikácií bežiacich na .NET Frameworku. Používa sa na vytváranie Windows desktopových aplikácií, webových služieb, klient-server aplikácií, databázových aplikácií a množstvo ďalších [4] .

3.3 WPF

Window Presentation Foundation (WPF) umožňuje vytvárať desktopové aplikácie pre operačné systémy Windows. Jadro WPF je renderovací engine nezávislý od rozlíšenia, ktorý je postavený tak aby vyhovoval modernému grafickému hardwaru. WPF prináša komplexnú sadu funkcií pre vývoj aplikácií, medzi ktoré patrí jazyk XAML (Extensible Application Markup Language), ovládacie prvky, data binding, layout, 2D a 3D grafika, animácie, šablóny a množstvo ďalšieho. WPF je súčasťou .NET Frameworku, čo zabezpečuje možnosť používať ďalšie prvky knižnice .NET Frameworku [6].

3.4 MVVM

Model-View-ViewModel (MVVM) je návrhový vzor vytvorený najmä pre WPF aplikácie. Ponúka riešenie ako oddeliť logiku aplikácie od užívateľského rozhrania. MVVM oddeľuje dáta, stav aplikácie a užívateľského rozhrania. Je tvorený troma vrstvami [7].

1. **Model** popisuje dáta s ktorými aplikácia pracuje.
2. **View** reprezentuje užívateľské rozhranie v jazyku XAML.
3. **ViewModel** spája Model a View a drží stav aplikácie.

Hlavný dôvod pre používanie MVVM vzoru je, že umožňuje viazanie dát (data binding) vo WPF aplikáciach.

3.5 Git

Git je distribuovaný systém na správu verzií. Bol vytvorený Linusom Torvaldsom v roku 2005. Pôvodne bol určený pre správu jadra operačného systému Linux. Jeho výhodou je, že každý používateľ má vlastný repozitár, v ktorom môže robiť lokálne zmeny. Tieto zmeny môže následne zosynchronizovať so serverom. Pri riešení konfliktov medzi lokálnym repozitárom a serverom je použitý trojcestný zlučovací algoritmus [5].

3.6 SQLite

SQLite je knižnica v jazyku C, ktorá implementuje malý, rýchly, nezávislý relačný databázový systém. SQLite je najpoužívanejší databázový systém na svete, je zabudovaný do všetkých mobilných telefónov a väčšiny počítačov a je dodávaný v nespočetnom množstve ďalších aplikácií, ktoré ľudia používajú na dennej báze [8].

4 Popis vlastnej práce

V tejto kapitole sa pokúsim rozobrať niektoré zaujímavé úlohy, ktoré som počas praxe riešil. Keďže som pracoval v agilnom tíme, častokrát som programoval časti úloh, ktoré samé o sebe nedávajú zmysel. Na základe toho som spojil niektoré úlohy do väčších celkov. Taktiež budem podrobnejšie písať len o tých častiach úloh, v ktorých som narazil na nové veci a rozširovali sa moje vedomosti.

V tejto práci nebudem mať priestor hlbšie sa venovať opravovaniu chýb z produkcie. Z toho dôvodu sa teraz pokúsim pár vetami zhrnúť túto problematiku. Počas mojej praxe som venoval bug fixingu približne 10-15% času mojej práce, čo je zhruba 12 dní. Tieto úlohy boli z môjho pohľadu najnáročnejšie, pretože boli zväčša z častí aplikácie s ktorými som nemal skúsenosti. A preto som mnoho krát musel najprv zistiť, čo daná funkcionalita vlastne robí, potom som musel preštudovať stávajúci kód a pomocou debugovania sa pokúsiť nájsť chybu. Keďže aplikáciu používa vyše tisíc ľudí z celého sveta na dennej bázy, párkrát sa stávalo, že chyba sa nedala vôbec reprodukovať. Takže na pár chybách som strávil pol dňa a nevyriešil ich. Na druhú stranu niektoré chyby sa dali nasimulovať a jednoducho opraviť. Z môjho pohľadu mi tento typ úloh priniesol možnosti ako viac spoznávať celú aplikáciu. Taktiež sa na základe týchto skúseností snažím predchádzať takýmto typom chýb pri písaní nového kódu.

Ďalšou časťou mojej práce bolo vytváranie unit testov. Na týchto úlohách som strávil dokopy približne 10 dní. Unit testy som zväčša písal pre metódy, ktoré som implementoval. V menšej časti som pokrýval už existujúce časti kódu unit testami. V týchto úlohach som riešil mnohokrát mockovanie objektov. Pri tom som si vyskúšal mockovanie metód pre rôzne vstupné parametre, taktiež všeobecne mockovanie metód a vlastností.

4.1 Zaznamenávanie používania a informácii potrebných pre opravy chýb

Časová náročnos: 14 dni
Prvá časť úlohy bola navrhnutie nového spôsobu monitorovania použitia jednotlivých funkcionalít aplikácie. Nová implementácia mala vyriešiť problémy pri behu viacerých inštancií naraz, rozdelenie dát podľa serveru, ku ktorému sa používateľ prihlásil, problémy so stratou dát pri nečakanom ukončení a taktiež nestabilné sieťové pripojenie. Časom sa vytvorila požiadavka na posielanie niektorých logov aplikácie na GrayLog a monitorovanie rýchlosti a stability sieťového pripojenia, pri ktorých sa javilo ako rozumné použiť podobný návrh riešenia. Ďalšia požiadavka bola monitorovanie rýchlosti a stability pripojenia aplikácie k serveru, kde sa využil taktiež rovnaký návrh.

4.1.1 Monitorovanie použitia

Keďže táto úloha bola už dlhšiu dobu v pláne, tak už bola urobená predbežná analýza problému a taktiež koncept návrhu nového riešenia . V prvom kroku mi kolega predal tieto informácie

a pokúsil sa mi načrtnúť ako by mohol vyzerať nový návrh. Tento návrh som pozostával vo vytvorení hlavnej triedy (Monitor), ktorá mala obsahovať metódu na pridávanie nového záznamu. Funkcia je volaná zvonku a dostane ako parameter nový záznam, ktorý bezpečne spracuje. Ďalej mala obsahovať frontu, do ktorej sa budú vkladať jednotlivé záznamy. Na triede mali bežať dve asynchrónne úlohy, jedna mala spracovávať položky z fronty a ukladať do súboru, druhá sa mala starať o posielanie súborov na server.

Pretože monitor sa mal získavať cez dependency injection, bolo potrene začať vytvorením rozhrania. Pri implementácií fronty som použil ConcurrentQueue, ktorá zabezpečuje, že ju môže používať viacero vlákien naraz. Metóda QueueConsumer zabezpečujúca spracovanie fronty, beží asynchrónne v cykle, až pokým sa nezavolá ukončenie monitoru. Pri ukončení sa všetky záznamy z fronty uložia do súboru, ktorý sa odošle pri neskoršom spustení aplikácie, respektíve ho môže odoslať aj iná inštancia, ktorá už beží. Pri tejto metóde bolo treba zabezpečiť aby sa nepokúšala stále v cykle vyťahovať záznamy z fronty ale aby počkala na prebudenie. Ako najlepšie riešenie sa javilo využiť konštrukciu AutoResetEvent, ktorú ponúka C#. Toto prebudenie sa vyvoláva pri vložení nového záznamu. QueueConsumer odoberie postupne všetky prvky z fronty a predá ich metóde ProcessInputItem, ak je fronta prázdna, úloha sa uspí. Metóda ProccessInputItem najprv načíta záznamy zo súboru, pridá nový záznam a uloží súbor. Pre jednoduché spracovávanie dát sú záznamy uložené v JSON formáte. Pri dosiahnutí určitého počtu záznamov sa súbor premenuje a prebudí sa úloha na spracovanie súborov, ktorá taktiež beží asynchrónne v cykle a používa AutoResetEvent. Pri písaní tejto metódy som musel vyriešiť problém aby sa viacej inštancií aplikácie nepokúsilo naraz posielať rovnaké súbory. Ako riešenie som zvolil mutexové zamykanie celej zložky so súbormi na odoslanie. Hlavná úloha tejto metódy je poslať dáta na server a v prípade neúspešného prenosu uložiť dáta na disk a pokúsiť sa poslať neskôr. Niektoré časti kódu, ako napríklad generovanie názvov súborov, som prevzal zo staršej implementácie. Na záver, po zavedení a otestovaní nového návrhu, som odstránil nepotrebné časti starej implementácie.

4.1.2 Logovanie do GrayLogu

Úloha pozostáva z pridania podpory na posielanie logov aplikácie na GrayLog. Podmienkou je zabezpečiť čo najspoľahlivejšie posielanie aj v prípade nekvalitného internetového spojenia.

Ako prvé som musel analyzovať možnosti posielania dát na Graylog. Tento systém dokáže prijímať dáta cez UDP, TCP alebo HTTP prokol. UDP som hneď vylúčil, kvôli nespoľahlivosti. Pri rozhodovaní medzi TCP a HTTP zavážilo najmä to, že cez TCP je možne posielať viac logov naraz, čo dokáže zmenšiť množstvo prenášaných dát. Keďže sme sa rozhodli použiť systém na spracovávanie logov z UsageMonitoru, ktorý vytvára súbory s viacerými logmi, je ako ideálne riešenie využiť TCP spojenie.

Na začiatku som vytvoril generickú abstraktnú triedu BaseMonitor. Táto trieda implementuje všetky spoločné časti pre UsageMonitor a GrayLogMonitor, ktoré z nej dedili. Rozdielne metódy ako napríklad odosielanie dát na server alebo spracovávanie nových dát je v každej triede

zvlášť. GrayLogMonitor spracováva dáta a v dávkach posiela na server. Pri posielaní sa vyžíva trieda GrayLogTcpClient, ktorá zaobaľuje konverziu správ do špeciálneho GELF formátu, nadviazanie spojenia so serverom a odosielanie dát. Nadviazanie spojenia je riešene asynchrónne a ak sa spojenie nenadviaže do troch sekúnd metóda vyhodí výnimku, ktorú odchytí GrayLogMonitor. V takomto prípade GrayLogMonitor uloží súbor na disk a pokúsi sa poslať neskôr. Pri tejto úlohe som narazil na problém s deserializáciou niektorých objektov z JSON súboru. Problém bol v tom, že niektoré triedy nemali prázdny konštruktor, ktorý využíva JSON deserializátor. Ako riešenie som zvolil prevádzanie dát na GELF formát, ktorý zabezpečí funkčnosť aj v prípade ak niektoré výnimky nebudú mať prázdny konštruktor. Ďalej som v tejto triede vytvoril časovač, ktorý v prípade ak sa za posledné tri minúty nenaplní dávka a nepošlú sa žiadne logy na server tak zabezpečí poslanie aj menšieho počtu logov.

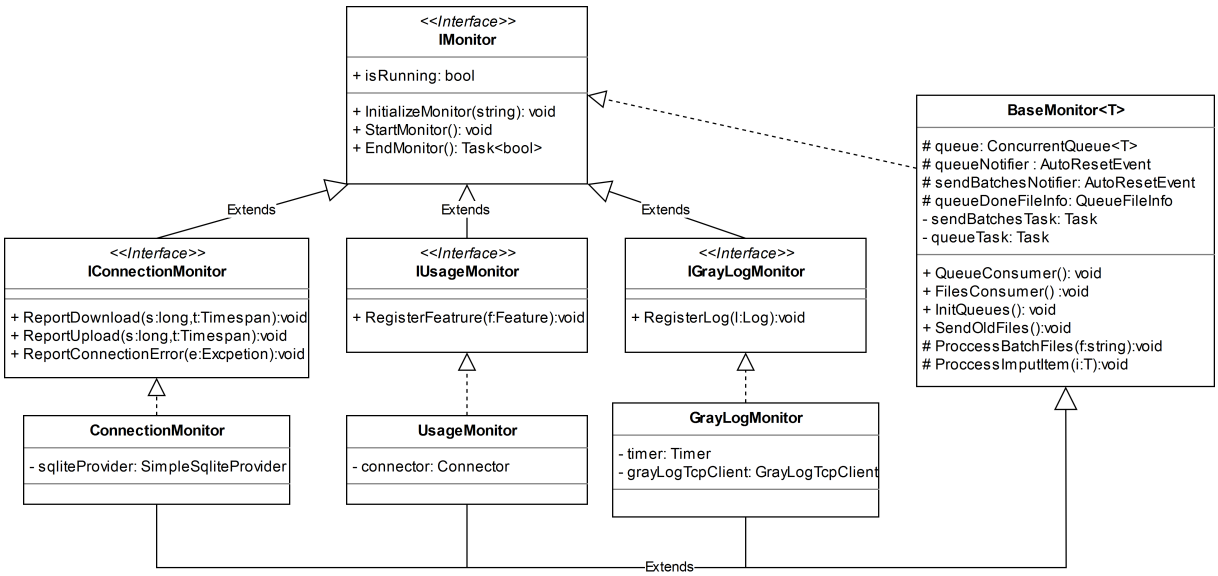
4.1.3 Monitorovanie rýchlosti a stability sieťového pripojenia

Úloha bola zameraná na zber informácií o rýchlosti a kvalite internetového pripojenia na neskoršie spracovanie. Požiadavka bola aby sa informácie od každej verzií aplikácie ukladali do jedného súboru, ktorý sa neskôr bude používať na určovanie kvality aktuálneho pripojenia. Na základe týchto informácií sa javilo ako rozumná voľba použiť základný koncept BaseMonitoru z predchádzajúcich úloh. Pri analýze spôsobu ukladania dát zvíťazila SQLite databáza pre rýchle a jednoduché získavanie informácií zo súboru.

Pri tejto úlohe som začal vytvorením dohodnutého rozhrania, aby sa mohlo súbežne pracovať na ostatných úlohách. Nasledovalo vytvorenie ConnectionMonitoru, ktorý rozširuje BaseMonitor. Konečný triedny diagram je priložený na obrázky A. Keďže systém bezpečného spracovania a zapisovania je vytvorený už v BaseMoniotoru, tak sa najzaujímavejšou časťou stala práca s SQLite databázou. Na prácu s databázou som použil už hotovú triedu, cez ktorú som otvoril databázu a potom cez metódu vyvolával SQL príkazy na vytvorenie tabuliek a vkladanie záznamov. Databáza obsahuje tri tabuľky: download, upload a connectionError. Informácie o rýchlosti a kvalite pripojenia, a taktiež či je používateľ pripojený cez VPN alebo priamo vo firemnej sieti, sa posielajú na Graylog. Informáciu o VPN pripojení som musel získavať zo sieťových adaptérov. Počas vývoja pribudla požiadavka na udržiavanie dát len posledných 30 dní. Na základe dobre zvoleného spôsobu ukladania dát do SQLite databázy to bola veľmi jednoduchá požiadavka. Na záver som vytvoril Unit test na funkčnosť ConnectionMonitoru. V tomto teste sa vytvára nová inštancia triedy, na ktorú sa volajú metódy na reportovanie dát a na záver sa porovnajú počty záznamov v tabuľkách s počtami vložených záznamov.

Zhruba po mesiaci nasadenia funkcionality sa objavila požiadavka na rozšírenie zaznamenávania informácii o sieťovom pripojení. Informácie mali zahŕňať spôsob pripojenia na sieť, dostupnosť verejného internetu a dostupnosť na privátny server. Pri tejto úlohe bolo najzložitejšie zanalyzovať dostupné riešenia a vybrať to najlepšie. Na získanie spôsobu pripojenia používam triedu NetworkInterface, ktorá má metódu GetAllNetworkInterfaces. Metóda vráti všetky sieťové adaptéry. Z týchto adaptérov sú vytiahnuté len zapnuté. Každý adaptér má v sebe vlastnosť

NetworkInterfaceType, na základe ktorej sa určuje typ pripojenia. Problematickejšie bolo nájsť riešenie na zistenie dostupnosti internetu a serveru. Zo začiatku som zvažoval možnosť pingovania ale táto možnosť nebola vhodná, kvôli problému s vybraním vhodného serveru. Ako rozumnejšie sa javilo použiť rovnaký systém ako používa Windows. Windows umožňuje cez rozhranie NetworkListManager použiť metódu IsConnectedToInternet, ktorá zabezpečí overenie pripojenia. Dostupnosť serveru som chcel riešiť pingovaním ale kolega mi poradil skúsiť využiť RabbitMQ, ktorý ma nadviazané spojenie so serverom. Rabbit ma na sebe vlastnosť IsOpen. Túto vlasnosť bolo potrebné propagovať do triedy ConnectionMonitor. Ako prvé riešenie som použil vytiahnutie objektu RabbitConnector v ConnetionMononitoru, kde som po spustení zistil, že aplikácia vyhadzuje výnimku cyklického vytvárania objektov. Pred týmto problémom som si neuvedomil, že RabbitConnector si vyťahuje ConnectionMonitor z MEF (kompozičná vrstva pre .NET, prinašajúca výhody voľne prepojenej architektúry) aby mohol reportovať štatistiky. Alternatívne riešenie som postavil tak, že ConnectionMonitor má vlastnosť typu Func, ktorej sa predá ako delegát metóda zisťujúca stav pripojenia. Posledná časť úlohy bola vytvoriť cestu na získanie SQLite súborov od uživateľov bez ich zásahu. Riešene je postavené na získaní UserDefinition (nastavenie pre každého používateľa) zo serveru, kde je bool vlastnosť, ktorá sa pri požiadavke získať dáta nastaví na true. ConnectionMonitor v metóde StartMonitor získa toto nastavenie a na základe nastavenej hodnoty sa rozhodne, či pošle dáta na server. Dáta sa posielajú na REST API adresu.



Obr. 1: UML diagram zachytávajúci kompozíciu tried monitoru

4.2 Vytváranie uživateľských rozhraní

Časová náročnosť: 18 dni

V priebehu mojej praxe vo firme prichádzali rôzne požiadavky na vytvorenie alebo úpravu užívateľských rozhraní, ktoré boli zväčša viac krokové wizardy alebo jednoduché okná.

Medzi moje prvé úlohy patrilo vytvorenie dvoch krokov v 4 krokovom wizarde. Keďže to bola jedná z mojich prvých skúseností s WPF aplikáciami musel som si na začiatku ozrejmiť návrhový vzor MVVM (Model, View, ViewModel), ktorý som pri vytváraní jednotlivých krokov potreboval ovládať. Ďalej som sa musel zoznámiť s návrhom wizardov, ktoré sa používajú v aplikácií. Pri riešení som vytváral dve stránky podľa wireframu. Prvá stránka obsahuje len text, comboboxy a textboxy. V tomto kroku som načítal model z predchádzajúceho kroku a pri prechode na ďalšiu stránku som uložil vybrané dáta do modelu, a poslal ďalej. Druhá stránka obsahuje všetky zvolené informácie z predchádzajúcich krokov. Tieto informácie som vytiahol z modelu, spracoval do zadanej textovej podoby a vytvoril prislúchajúci pohľad. Súčasťou tejto úlohy bolo aj zaistenie validácie a v prípade splnených podmienok odoslanie požiadaviek na server.

Ďalšiu úlohou bolo vytvorenie okna, v ktorom si používateľ navolí filter. Pri realizácií som použil už tri hotové komponenty. Pre tieto komponenty bolo potrebné vytvoriť odpovedajúci ViewModel, na ktorého property sa nabindovali časti komponent. Toto okno pri zatvorení posiela request na server. ViewModel taktiež obsahuje validačnú metódu. Validačná metóda vyhodnotí, či je korektne zvolený filter, v prípade nejakých nedostatkov sa objavia hlášky vo validačnej komponente. Pre otváranie okna som pridával nové tlačidlo a klávesovú skratku.

Moja nasledujúca práca bola vytvorenie dvojkrokového wizardu. Pri implementácií som vytvoril triedu dediacu zo základného WizardViewModelu. Táto trieda je kostra celého wizardu. Obsahuje metódy na inicializáciu jednotlivých krokov, prechody medzi stránkami taktiež metódy, ktoré sú vyvolané pri ukončení wizardu. Súčasne som vytvoril dve stránky. Každá z týchto stránok ma dve triedy, view a viewmodel. Na jednej stránke bola požiadavka aby sa jednotlivé časti zobrazovali podľa zvolených informácií. Požiadavku som vyriešil priradením vlastnosti viditeľnosť na bool propertu vo viewmodeli, ktorá sa nastavovala podľa zvolených dát. Kvôli rôznym dátovým typom som použil základný konvertor BoolToVisibility.

Pri rozširovaní funkcionality wizardu o pridanie nového typu pravidla som musel analyzovať už vytvorenú časť kódu. V pohľade sa nachádzal listbox, do ktorého bolo potrebné vložiť novú položku. Táto nová položka mala obsahovať ďalší listbox. Problém bol v tom, že položky boli zaobalené triedou SimplePartOption a pre vykresľovanie sa používala templata vytvorená pre túto triedu. Kvôli zachovaniu existujúceho kódu som vytvoril novú triedu FunctionPartOption dediacu zo SimplePartOption. FunctionPartOption obsahuje list objektov základnej triedy. Toto riešenie mi umožnilo pridanie novej položky do listu. Ďalej som musel vytroviť novú templatu pre vnorený listbox a zabezpečiť správny výber templaty. Pre výber templaty som vytvoril selector, ktorý rozširuje základný DataTemplateSelector.

```
  <HierarchicalDataTemplate x:Key="AddQcFunctionPartOptionTemplate" DataType="{
     x:Type viewModels:AddQcFunctionPartOption}">
<toolBar:ToolbarItem HeaderOrientation="Horizontal" HeaderText="{Binding
    Caption}">
<ItemsControl ItemsSource="{Binding AddQcSimplePartOptions, RelativeSource={
    RelativeSource AncestorType={x:Type TreeViewItem}}}"
AutomationProperties.AutomationId="QcSectionPart_OptionsItemControl"
    ItemTemplate="{StaticResource AddQcSimplePartOptionTemplate}">
</ItemsControl>
</toolBar:ToolbarItem>
</HierarchicalDataTemplate>
```

Výpis 1: Dátová templata pre triedu AddQcFunctionPartOption

Nasledujúca úloha bola vytvoriť jednu stránku, na ktorej boli 4 radiobuttony a podľa toho, ktorú možnosť užívateľ zvolil sa zobrazili rozširujúce možnosti. Jedna z možnosti otvorí nové okno s filtrom. Tento istý typ okna s malými zmenami sa používal na troch miestach v aplikácií a všade malo vlastnú implementáciu. Z toho dôvodu som mal za úlohu vytvoriť z neho novú WPF komponentu, ktorá sa bude môcť používať ako ostatné komponenty. Keďže táto nová komponenta obsahuje validačnú komponentu a v každom prípade sa používa iná validačná metóda, tak som musel riešiť problém ako predať správnu metódu. Ako riešenie som použil property typu ReturnRelayComand, ktorá bola zároveň priradená ako DepencencyProperta.

```
public ReturnRelayCommand<object, Dictionary<string, ValidationState>>
    FilterValidationCommand
{
   get => (ReturnRelayCommand<object, Dictionary<string, ValidationState>>)
       GetValue(FilterValidationCommandProperty);
   set => SetValue(FilterValidationCommandProperty, value);
}
public static readonly DependencyProperty FilterValidationCommandProperty =
    DependencyProperty.Register("IsFilterValidCommnad",
   typeof(ReturnRelayCommand<object, Dictionary<string, ValidationState>>),
typeof(FilterItemSelectorComponent), null);
```

Výpis 2: ReturnRelayComand na predanie validačnej metódy

Pri implementácii úlohy, ktorá rozširuje možnosti pre užívateľov som musel vyriešiť aby sa určité časti na view zobrazovali podľa zvolenej možnosti v comboboxe. Ako riešenie som vytvoril konvertor. Tento konvertor ma propertu enums, kde sú nastavené hodnoty, pre ktoré vracia

visible. Pre ostatné hodnoty vracia hidden. Konvertoru sa vlastnosti nastavujú priamo z XAML súboru.

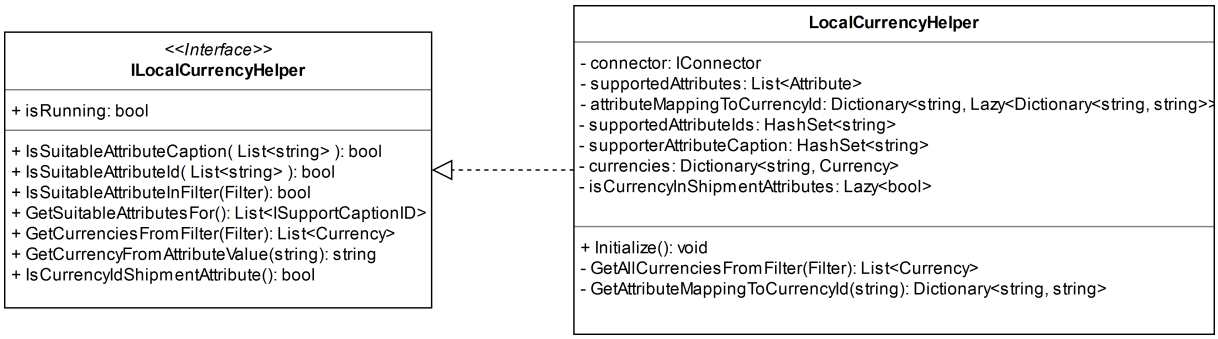
```
<converters:MultiEnumsEqualsToBoolConverter x:Key="
    MultiEnumsEqualsToBoolConverter">
<converters:MultiEnumsEqualsToBoolConverter.Enums>
<simple:QcRuleOperator>CANNOT_BE_CALCULATED</simple:QcRuleOperator>
<simple:QcRuleOperator>CAN_BE_CALCULATED</simple:QcRuleOperator>
</converters:MultiEnumsEqualsToBoolConverter.Enums>
</converters:MultiEnumsEqualsToBoolConverter>
```

Výpis 3: Nastavenie vlastností konvertoru z XAML súboru

4.3 Rozšírená podpora pre lokálne meny

Časová náročnosť: 6 dni

Doteraz sa pracovalo v aplikácii s lokálnou menou na základe krajiny. Pribudla požiadavka aby bolo možné pracovať v jednej krajine s rôznymi menami. Keďže táto zmena zasahovala do viacerých častí aplikácie ako sú importovanie, exportovanie a zobrazovanie dát, bolo potrebné vytvoriť jednu triedu, ktorá pokryje všetky metódy na validáciu dát ohľadom lokálnej meny. Keďže implementácia tejto úlohy zasahuje aj do funkcionalít, ktoré sa navonok nemenia, bolo potrebné vytvoriť integračný test. Tento test zabezpečí, že po zmenách v kóde sa zaručí nezmenená funkcionalita. Pri vytváraní týchto testov som použil už jestvujúci nástroj na nahrávanie zmien v aplikácií. Nástroj uloží do súboru pôvodné dáta, všetky užívateľské zmeny a výsledné dáta. Následne som tieto súbory vložil k ostatným integračným testom.



Obr. 2: Rozhranie a trieda pre lokálnu menu

Táto trieda implementuje rozhranie ILocalCurrency, ktoré obsahuje všetky potrebné metódy na prácu s lokálnou menou. Návrh rozhrania a triedy môžete vidieť na obrázku 2. Pri vytváraní fieldu geographyMap som použil Lazy konštrukciu, ktorá je založená na návrhovom vzore Lazy Loading. Je to z dôvodu, že získanie dát na vyhodnotenie metódy zaberie určitý čas a niektoré

inštancie triedy vôbec nepoužívajú tento field, a zároveň v niektorých metódach sa field používa v cykloch.

```
geographyMap = new Lazy<DimensionMap>(() => connector.GetItem<DimensionMap>(
    technology.CurrentTechnology.ShipmentMetadata.GeographyMapId, technology.Id
    ));
```

Výpis 4: Vlastnosť na získanie geografickej mapy využívajúca oneskorenú inicializáciu

Ďalšia zaujímavá metóda je GetAllCurrenciesFromFilter, ktorá vezme Filter vytiahne z neho prienik všetkých lokálnych mien. Metóda postupne prechádzala všetky položky vo filtri a vyťahovala z nich meny. Problém bol v tom, že filter mohol obsahovať položku Country, v ktorej boli zvolené nejaké krajiny, a zároveň položku CountryID, v ktorej boli zvolené ID iných krajín. Na vytváranie preniku som použil HashSet stringov, ktorý má metódu IntersectWith.

```
var hashSet = new HashSet<string>();
if (currenciesList.FirstOrDefault() != null)
{
hashSet.AddRange(currenciesList.FirstOrDefault());
foreach (var currency in currenciesList.Skip(1))
   hashSet.IntersectWith(currency);
}
```

Výpis 5: Použitie HashSetu na získanie prieniku lokálnych mien

Po úpravách na importe a exporte som pri validácii používal objekt triedy LocalCurrencyHelper. Na základe čoho sa zmenili niektoré rozhrania. Po týchto zmenách začalo padať pár unitových testov, ktoré som postupne opravil.

4.4 Resetovanie Cache

Časová náročnosť: 2 dni
Táto úloha zahŕňa vytvorenie tlačidla pre resetovanie cache aplikácie (stiahnutie zo serveru), ktoré zároveň vyvolá znovu vytvorenie cache na serveri. Táto možnosť má byť dostupná len užívateľom s určitými právami. Na začiatku som vytvoril novú UserControl , ktorá reprezentuje tlačidlo evokujúce resetovanie cache. Túto UserControl som pridal do MEFu, pretože tlačidlo je registrované cez Prism Region (framework, ktorý používa kontajnere na vytváranie štruktúrovaného a dynamického UI), ktorý z neho vyťahuje objekty. V konštruktore sa vyťahuje EventAgregator. EventAggregator je knižnica, ktorá umožňuje komunikáciu medzi voľne viazanými komponentami bez priamej referencie. Tento EventAggregator je použitý na publishnutie ForceResetCache eventy po stlačení tlačidla. V CacheManageri je táto eventa subscribnuta.

```
private async void FlushCacheMenuItemView_OnClick(object sender,
    RoutedEventArgs e)
{
//run flush cache asynchronously
await Task.Run(() => eventAggregator.GetEvent<FlushCacheEvent>().Publish(new
    FlushCacheEventAgrs()));
}
eventAggregator.GetEvent<FlushCacheEvent>().Subscribe(ResetCache);
```

Výpis 6: Odoberanie udalosti zo služby zabezpečujúcej komunikáciu medzi komponentami

Táto koncepcia zabezpečuje, že po stlačení tlačidla sa vyvolá v CacheManageri ResetCache metóda. V ResetCache metóde je umiestnená logika zabezpečujúca odosielanie požiadavku na server a po prijatí odpovede prepísanie cache súboru.

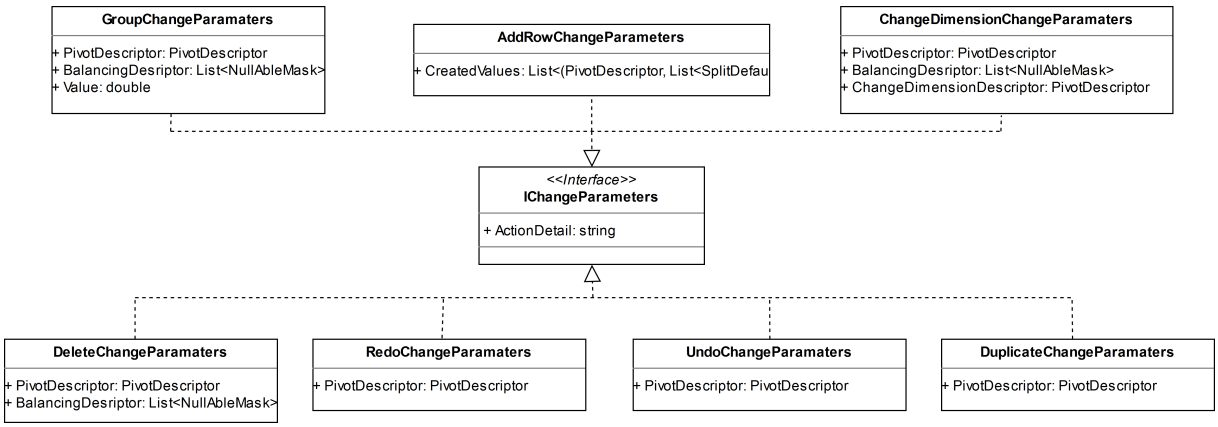
Druhá časť bola zabezpečiť viditeľnosť tlačidla len pre uživateľov, ktorý majú dané právo pre technológiu. Pri riešení tohto problému som ako prvotné riešenie zvolil, že sa v konštruktore tlačidla subscribne TechnologyChangeEvent, ktorá pri výbere alebo zmene technológie vyvolá eventu s parametrom názvu vybranej technológie. Z MEFu sa vytiahne UserRightsManager, kde je metóda, čo na základe názvu technológie zistí, či ma užívateľ potrebné práva. V závislosti od výsledku metódy sa mení properta visibility tlačidla. Akonáhle som skúsil otestovať toto riešenie, zistil som, že je nefunkčné. Problém bol v tom, že objekt tlačidla sa vytvára až po kliknutí na menu a kliknutie na menu prichádza až po výbere technológie. Čiže TechnologyChangeEvent je vyvolaná ešte pred subscribnutím tejto eventy objektom. Tento problém som vyriešil vytvorením ResetCacheProvideru. Provider ma propertu, ktorá reprezentuje viditeľnosť tlačidla a je uložený v MEF ako singleton. Provider v konštruktore subscribuje TechnologyChangeEventu a priradzuje k nej metódu, ktorá získa práva užívateľa pre technológiu a na základe toho nastaví propertu viditeľnosti. Prvotná inicializácia provideru je vykonaná pri štarte aplikácie. Keďže provider je singleton, tak ho môžem v konštruktore reset cache tlačidla vytiahnuť z MEF a na základe jeho property nastaviť viditeľnosť tlačidla.

4.5 Rozšírenie podpory integračných testov

Časová náročnosť: 4 dni

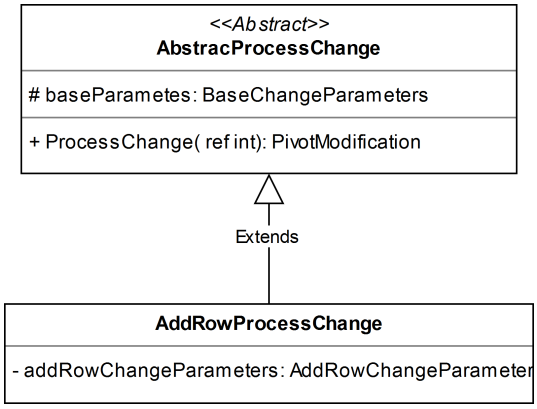
Prvotnou úlohou bolo vytvoriť integračný test, ktorý pokryje novú funkcionalitu aplikácie. Na vytvorenie testu som používal už jestvujúci nástroj na nahrávanie zmien v aplikácií. Nástroj uloží do súboru pôvodné dáta, všetky užívateľské zmeny a výsledné dáta. Kvôli novej funkcionalite, ktorá vytvára nové dáta bolo potrebné rozšíriť jestvujúce framework o nové možnosti. Potrebné bolo najmä zabezpečiť aby framework dokázal spracovávať nový typ zmeny. Tento framework generuje a spracúvava excel súbory. Každý súbor reprezentujú tri stránky. Na prvej stránke sú pôvodné dáta na druhej sú zmeny, ktoré sa majú nad dátami vykonať a na poslednej

sa nachádzajú dáta vo výslednej podobe. Pre novú funkcionalitu test zvláda vytvoriť stránku s pôvodnými a výslednými dátami. Problém je so zaznamenaním zmeny. Keďže cieľom tejto úlohy nebolo zabezpečiť nahrávanie testov ale vytvoriť testy, tak som sa zameral len na spracovanie súboru s novou zmenou. Aby bol možné spracovať nový typ zmeny musel som najprv rozšíriť funkciu, ktorá načíta zmeny z excelu. Pri realizovaní tejto zmeny som narazil na problém, že metóda vracala List trojice objektov a ďalej mala 4 vystupné parametre. Pre zabezpečenie spracovania nového typu zmeny by som musel pridať 5 výstupný parameter a vložiť ďalšie podmienky do metódy, ktorá bola už pred týmito zmenami zložito čitateľná. Taktiež sa ďalej pri spracovávaní zmien používajú všetky výstupné parametre a neprehľadné vetvenie. Na základe týchto zistení som sa rozhodol zmeniť štruktúru týchto metód. Navrhol som riešenie, v ktorom metóda získavajúca zmeny z Excelu vracia priamo list objektov implementujúci rozhranie IChangeParameter. V tejto metóde sa prechádzajú postupne všetky riadky súboru, kde každý riadok reprezentuje jednu zmenu. V prvom stĺpci sa nachádza názov a detail zmeny. Na základe názvu zmeny sa vytvára objekt pre konkrétny typ zmeny. Tento objekt si načíta len všetky potrebné údaje. Týmto som zabezpečil, že pri ručnom vytváraní zmien stačí vypĺňať len potrebné stĺpce v tabuľke, čo bolo v predchádzajúcom riešení nedokonale.



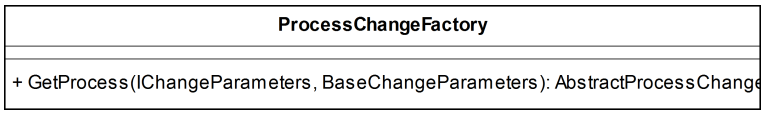
Obr. 3: Triedny diagram pre parametre testov

Pre spracovanie zmien som navrhol riešenie, v ktorom je pre každý typ zmeny samostatná trieda. Tieto triedy rozšírujú abstraktnú triedu AbstractChangeProcess a implementujú metódu ProcessChange. Táto metóda berie ako parameter index aktuálnej zmeny, pretože niektoré zmeny môžu vykonávať viacero akcií naraz, a vracia modifikáciu. Ďalej každá konkrétna trieda obsahuje field baseParameters, kde sú všeobecné údaje, a druhý field, ktorý obsahuje ostatné potrebne parametre.



Obr. 4: Návrh procesu na spracovanie zmien

Na získavanie správneho objektu som vytvoril statickú triedu ProcessChangeFactory. Táto trieda má metodú GetProcessChange, ktorá obsahuje switch, kde sa na základe typu parametru vytvorí objekt. Vytvorenému objektu predá potrebné údaje a vráti ho.



Obr. 5: Továreň na získavanie objektu na základe typu vstupných parametrov

V tomto navrhnutom riešení stačí pri spracovávaní zmien v testových súborov len zavolať metódu, ktorá vráti list parametrov zmien. Tento list sa preiteruje a pre každú položku sa získa z továrne objekt, na ktorý sa zavolá metóda ProcessChange. Výsledná implementácia je mnohonásobne prehľadnejšia, čo sa v budúcnosti odrazí rýchlejším vytváraním nových testov.

4.6 Zmena dimenzie

Časová náročnosť: 3 dni
Hlavná úloha pozostávala z pridania novej funkcionality, vďaka ktorej mohol používateľ meniť dimenziu jednotlivých shipmentov (najmenšia jednotka na uchovávanie dát v aplikácii, ktorá obsahuje dimenzie a hodnoty). Moja časť úlohy sa skladala z vytvorenia commandu, ktorý prijíma dané parametre, validácie, či sa môže command vykonať, vytvorenie špeciálnej masky, ktorá sa predala ako parameter ďalšej metóde, postaranie sa o prekreslenie view a na záver vytvoriť unit test na otestovanie danej funkcionality.

Pri riešení úlohy som si najprv upresnil s tímom dané rozhrania, keďže sme súbežne traja pracovali na svojich častiach a tie sa navzájom prepojili. Ďalej som si rozplánoval postup riešenia. Ako prvé som sa zameral na napísanie funkcie, ktorá vytvára nullable masku. K tejto funkcií som mal vytvoriť unit test. Tento test sa neskôr ukázal ako najnáročnejšia časť zadania. Najväčší problém nastal pri mockovaní objektu, ktorý funkcia prijímala ako parameter. Keďže tento objekt mal zložitú štruktúru a veľa metód, ktoré boli potrebné, strávil som pritom najviac času. Na

druhú stranu som sa naučil mockovanie rôznych druhov metód a vlastností. Ďalej som vytvoril validáciu commandu. Pri riešení validácie som prebral s QA všetky možné scenáre kedy musí byť akcia zakázaná. Ako posledné som vytvoril daný command.

4.7 Pridanie nových riadkov/stĺpcov do pivotu

Časová náročnosť: 2 dni

Cieľom zadania bolo pridať novú hodnotu na spravnú pozíciu v hlavičkách. Táto úloha pozostávala z vytvorenia metódy, ktorú pokryje unit test, a zaistenia funkčnosti Modification trackru.

Ako prvé som si zanalyzoval všetky možne kombinácie, ktoré musí metóda pokryť. Potom som si vytvoril jednoduchú metódu na pridanie hodnoty na konkrétnu pozíciu v hlavičkách. Ďalej som vytvoril metódu, ktorá volala predchádzajúcu a predávala jej hodnoty a pozície. Keďže tých možností bolo mnoho, metóda musela byť komplexná. Pri postupnom testovaní som zistil, že pri analýze som pozabudol na niektoré okrajové prípady, preto som musel kód viackrát upravovať. Pri písaní unit testu som pokryl všetky prípady, pri ktorých mi prvotná implementácia nefungovala správne. Pre zaistenie správnej funkčnosti undo/redo som musel rozšíriť modification tracker o novú vlastnosť, ktorá zaznamenala údaje o novo pridaných riadkov alebo stĺpcov. Pri riešení úlohy som taktiež opravil dva drobné problémy, ktoré sa objavili pridaním novej funkcionality. Na záver som pridal integračný test, ktorý používa predpripravený framework. Tento test simuluje aplikáciu, v ktorej sa vyvolá metóda a na záver porovná výsledok aplikácie s pripraveným Excel súborom.

5 Zhodnotienie

Celkovú prax hodnotím veľmi pozitívne. Z môjho pohľadu som sa naučil veľa nových veci a hlavne mi prax priniesla nespočetné množstvo skúsenosti s vývojom aplikácií, pracou v tíme, komunikáciou, taktiež prehľad o celom cykle vývoja. Rozhodne by som nemal možnosť získať tieto cenné skúsenosti pri vytváraní bakalárskej práce. Preto hodnotím výber praxe ako správnu voľbu.

V priebehu praxe som mal možnosť využívať väčšinu vedomosti, ktoré som získal počas doterajšieho štúdia na vysokej škole. Už na vstupnom pohovore mi pomohli vedomosti, najmä z predmetov o programovacích jazykoch, k získaniu práce. Keďže som pracoval ako vývojár boli pre mňa prínosom všetky predmety o programovacích jazykoch. Na dennej bázy som používal vedomosti získane na predmetoch Algoritmy a Softwarové inžinierstvo. Pri riešení problémoch s blokovaním súborov a procesov mi veľmi pomohol predmet Operačné systémy. Taktiež veľmi užitočný bol základný prehľad o celkovom vývoji aplikácií, ktorý som získal na predmetoch DAIS a VIS. Pri práci z SQLlite databázou mi postačovala syntax, ktorú som si osvojil v predmete úvod do databázových systémov. Zároveň som počas praxe narazil na problémy, kde boli užitočné základné znalosti z počítačových sieti. Párkrát vytváral regulárne výrazy, kde som zúžitkoval informácie z UTI.

Na druhú stranu mi niektoré vedomosti chýbali. Napríklad aspoň základná znalosť o verzovacom systéme Git by mi určite uľahčila prvé dni v práci. Taktiež mi chýbali aspoň teoretické znalosti o agilných metodikách vývoja a spolupráci v tíme. Zároveň by som privítal pár prednášok o vývoji, testovaní a nasadení softvéru v praxi. Keďže v dnešnej dobe sa kladie veľký dôraz na unitové, integračné alebo UI testy, bolo by prínosné venovať sa tejto problematike aj v škole.

6 Záver

Absolvovanie odbornej praxe v IDC splnilo všetky moje očakávania. Prax mi umožnila zapojiť sa do vývoja aplikácie, ktorá denne pomáha tisícke analytikov, čo bolo omnoho motivujúcejšie ako pracovať na školských projektoch. Mal som možnosť zistiť ako prebieha kompletný cyklus vývoja aplikácií.

Naučil som sa pracovať v agilnom tím, kde je veľmi dôležitá komunikácia medzi jednotlivými členmi. Na základe rôznorodosti požiadaviek prichádzajúcich od analytikov cez product ownera, som si vyskúšal prácu na viacerých oblastiach aplikácie a najmä rôzne technológie. Vďaka tomu, že som pracoval na celkom rozsiahlej aplikácii, kládol sa veľký dôraz na kvalitu kódu. Ak pri code review skúsený kolegovia narazili na nejaký nedostatok v efektivite, čitateľnosti kódu, prípadne ak vedeli o lepšom riešení, tak som musel svoje riešenie prerobiť. Tento postup mi umožnil rýchlo si osvojiť skúsenosti, ktoré oni naberali počas celej svojej praxe. Počas práce vo firme som získaval každý deň nové vedomosti, učil sa používať nové technológie a zlepšoval moje schopnosti. Taktiež som mal možnosť zužitkovať moje doterajšie vedomosti získané počas štúdia. Overil som si vhodnosť výberu povolania a najmä som si zvýšil uplatniteľnosť na trhu práce.

Vo firme ostávam pracovať naďalej. Na základe voľnejšieho rozvrhu v škole a obojstrannej spokojnosti som začal pracovať od februára ako plnohodnotný člen tímu na plný úväzok.

Literatúra

[1] IDC. *IDC about.*[online]. [cit.31.3.2019].
Dostupné z: https://www.idc.com/about

[2] IDC. *Ferda.*[online]. [cit.31.3.2019].
Dostupné z: https://www.idc.com/cz/careers/projects

[3] Microsoft. *.NET Framework. What is .NET?* [online]. [cit.31.3.2019].
Dostupné z: https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet

[4] Microsoft. *Introduction to the C# Language and the .NET Framework.* [cit.31.3.2019].
Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework

[5] Jon Loeliger a Matthew McCullough.Version Control with Git: Powerful tools and techniques for collaborative software development. Ö'Reilly Media, Inc., 2012.

[6] Microsoft. *WPF overview.* [cit.31.3.2019].
Dostupné z: https://docs.microsoft.com/en-us/visualstudio/designers/introduction-to-wpf?view=vs-2017.

[7] DotNetPortal. *MVVM: Model-View-ViewModel.* [cit. 31.3.2019].
Dostupné z: https://www.dotnetportal.cz/clanek/4994/MVVM-Model-View-ViewModel

[8] SQLite. *What Is SQLite?* [cit.31.3.2019].
Dostupné z: https://www.sqlite.org/index.html